A simple navigation framework for the RASCAL robot

Arunkumar Byravan
Department of Mechanical Engineering
University of Pennsylvania

Abstract—**In this work, we explore a simple navigation framework for semi-autonomous robots in outdoor environments. Three main components were conceptualized and implemented. Accurate localization is achieved by fusing measurements from an onboard GPS, IMU and odometry. A LIDAR mounted on a tilting platform is used to generate a local map of the surrounding environments. The robot then uses this map to plan a path and navigate around obstacles. A Finite State Machine is developed to reflect the set of possible actions the robot can take. We present results from each of the individual components and evaluate their performance.**

I. Introduction

NASA's Revolutionary Aerospace Systems Concepts Academic Linkage (RASCAL) competition challenges teams to build a planetary rover for semi-autonomous space exploration. The rovers explore the environment, looking for objects of interest (colored rocks), which the rovers are supposed to collect. The rovers are expected to have some degree of automation, but they can be remotely tele-operated through a broadband card. The environment is mostly devoid of features and consists of mock up surfaces of the Moon, the Mars and a rock yard. An image of the testing site is shown in figure 1. To this end, this work aims to establish a robust framework for efficient navigation of the rover. The following sections explain each of the various components in greater detail. Section 2 talks about related work on localization and navigation through outdoor environments. Section 3 explains the localization module. Section 4 talks about the State Machine, Costmap generation and planning. Both these sections are self consistent and give results from various experiments on each of the individual modules. Finally, section 5 summarises the work and gives concluding remarks.

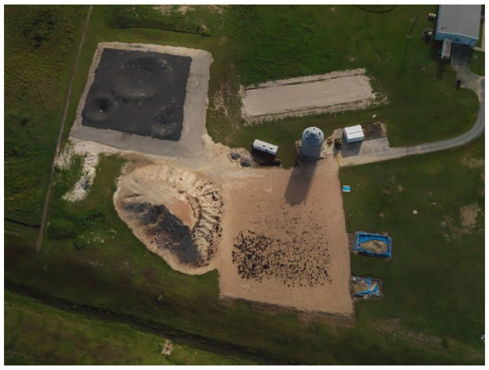
Fig. 1. Test site for NASA'S RASCAL competition

II. Related Work

The Kalman Filter is an estimation technique which can be used to track the state of the system even when the system model is not accurately known and there is a lot of noise in the system. Since its introduction, it has become one of the most widely used and robust techniques for estimating the states of noisy systems. The Kalman filter has also been applied to the problem of estimating a robot's 3D pose, both indoors and ourdoors. There has been a lot of research related to robot localization using a multitude of sensors. [1] gives an efficient algorithm for tracking the 3D pose outdoors using a low cost GPS, Inertial and Visual sensors. They use a multirate Extended Kalman Filter (EKF) to fuse observations from the various sensors. [4] talks about a Rao-Blackwellized particle filter based approach for 3D localization. Observations from GPS, a low cost IMU and Odometric sensors are integrated to give an accurate and drift-free pose estimate. The approach has many advantages compared to a simple Kalman Filter, but is also more computationally expensive. [2] describes an attitude filter based on the Unscented Transform and quaternions that gives very good results and avoids the singularities of a different rotation representation. It is the basis for the attitude filter used in our system. We now proceed to explain the concpetual and implementation specific details of out work.

III. Localization

Accurate localization of the robot's position over time is crucial to any exploration that the robot does. Our specific application entails us to operate in an outdoor environment devoid of features. The robot is equipped with a number of sensors, including an omni-directional camera, a GPS, an IMU and encoders. The estimation of the 3D pose of the robot can either be done jointly or the attitude can be estimated separately from the position. We have chosen to do the latter, wherein we use two separate UKF's, one for the estimation of the attitude and one for the estimate of the position and heading. The IMU is used for the purpose of the attitude estimation, while for the second UKF, a combination of IMU,GPS and Encoders are used.

A. Unscented Kalman Filter

The Unscented Kalman Filter is an extension of the general Kalman Filter to non-linear systems. Instead of linearizing the model (as the Extended Kalman Filter does), the distribution is transformed to a set of ”sigma” points through the Unscented Transform. These sigma points are then updated using the true non-linear model which results in a more accurate estimate than the EKF. Also, it is computationally effective and there is no need to evaluate any Jacobians which are needed for an EKF.

1) Why UKF?: As described before, we have two different filters, one for the attitude and another for the position. The attitude is parametrized as a quaternion and so the state is 4 diumensional. Rotations in general are a non-linear space and the process updates therefore are not strictly linear, which facilitates the use of the UKF. The attitude filter is very similar to the one described in [2] and will not be explained in greater detail in this work. Our focus is more on the position estimation and we will assume that the attitude is given. As for the position, it is parametrized as a 3 dimensional state $[X, Y, Heading]^T$. Due to lack of measurements in the IMU along the heading direction, the heading given by the attitude filter drifts over time and we have chosen to re-estimate this. In our application, we have assumed a differential drive model for the robot and so the process update depends non-linearly on the heading and the control input (encoders). This drives us to use a UKF.

2) Sensors: We have chosen to use a combination of IMU, GPS and encoders to achieve localization. Initially, we had tried using an omni-directional camera for estimates of the yaw. During testing on the actual rover, we found out that the robot vibrates and pitches a lot which induces blur and random motions in the omni-directional image. This along with computational constraints made the camera option unfavorable.

3) Filter Details: The implementation is based on the general UKF given in [3]. The state is a 3 dimensional vector of X and Y positions and Heading. The yaw rate from the IMU is used for the process udpate for the yaw. Encoder measurements are used as the process update for the X and Y position. These updates depend on the current heading and pitch and roll of the robot. The X and Y positions and the Heading from the GPS are used as measurement updates. The working of the filter is detailed in algorithm 1.

4) GPS/Encoder calibration: The Global Positioning System (GPS) computes the latitude, longitude and heading of a moving object based on triangulation using a number of satellites. The heading that we get from the GPS is with respect to a North - East - Down reference frame. Without knowing the robot’s orientation w.r.t this frame, measurements from the encoder(which are in the body frame) cannot be integrated with the GPS. A magnetometer can give the robot’s absolute orientation which can then be used to do the fusion. But, a magnetometer is easily affected by the environment and needs careful tuning which lead us not to use one in our setup. In order to perform this calibration, the robot initially waits till it gets a good fix on its position and then drives straight while accumulating GPS readings. It then computes the mean of the headings from the GPS. This mean heading is used as the initial orientation estimate of the robot.This step could be omitted and the robot’s initial orientation can be taken as zero, in which case the Kalman Filter would eventually converge to its actual orientation.

Algorithm 1 Unscented Kalman Filter for 2D state estimation

```
Calibrate GPS and Encoders to obtain initial orientation
Initialize State to $[0; 0; InitialOrientation]$
Initialize State covariance $R$ and noise
while New measurement do
  if IMU then
    Process update for heading
    $Sigma \Leftarrow chol(const * R + GyroProcessNoise)$
    $Sigma \Leftarrow Sigma + [0; 0; Yawrate * dt]$
    $State \Leftarrow mean(Sigma)$
    $Sigmams = Sigma - State$
    $R \Leftarrow (Sigmams) * (Sigmams)^T$
  else if Encoders then
    Process Update for X & Y
    Compute $dx$ & $dy$ using motion model
    $[dxw; dyw; dzw; 1] \Leftarrow R_BodytoWorld * [dx; dy; 0; 1]$
    $Sigma \Leftarrow chol(const * R + EncoderProcessNoise)$
    $Sigma \Leftarrow Sigma + [dxw; dyw; 0]$
    $State \Leftarrow mean(Sigma)$
    $Sigmams = Sigma - State$
    $R \Leftarrow (Sigmams) * (Sigmams)^T$
  else if GPS then
    Measurement update
    $Meas \Leftarrow [GPS_X; GPS_Y; GPS_Heading]$
    $Innov \Leftarrow Measurement - State$
    Compute Innovation covariance
    Compute Cross correlation and Kalman Gain
    Update State and State Covariance
  end if
end while
```

B. Results and Discussion

The system described above was implemented on a mobile robot and tested on datasets collected outdoors. Most of the data has been collected from areas which have an unobstructed and clear view of the sky. The robot was driven around using a joystick during the runs. The robot is equipped with a low cost inertial sensor and encoders. The attitude filter runs in a microcontroller and spits out estimates of the robot’s roll,pitch and yaw. Both the IMU and the attitude filter run at 100 Hz. The encoders run at 40 Hz. The robot also has a U-Blox EVK-6H GPS Evaluation kit which gives position and heading estimates at 5 Hz. The heading calculation is decoupled from that of the position. The GPS was tested beforehand and was found to have an accuracy of about 1 - 1.5 m on open ground with no buildings. Results from experiments on the

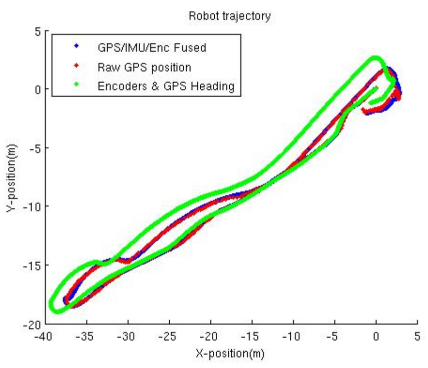

Fig. 2. Robot trajectories

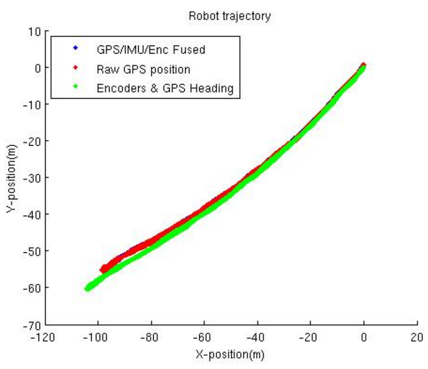

Fig. 3. Robot trajectories

localization module are shown in the figures below. Figures 2,3 & 4 show the trajectories taken by the robot. The red trajectory is the raw GPS value, the blue the result of the sensor fusion using the UKF and the green using the Encoders and the heading from the GPS alone. From the plots, we can see that the UKF does well in fusing the high frequency components (IMU & Encoders) with the low frequency estimate from the GPS to give a smooth trajectory. The system also does not drift a lot except in a few cases where we can see jagged edges in the trajectory. In these places, it can be seen that the heading lags the position from the GPS. Figures 2 and 3 have loop closures in them and it can be seen that the system is able to close the loop without a lot of error. The average error in all the cases was found to be less than 2 m which also shows the accuracy of the GPS readings. Due to ground truth not being available, detailed analysis of the accuracy of the system could not be done. An attempt was made to collect ground truth data by driving the robot on a straight line along the curb, but due to presence of builds and trees in the vicinity, the GPS readings showed a lot of errors.

C. *Improvements*

There are a number of improvements that can be made to the above mentioned system. A sensor such as the magnetometer that gives the yaw value more accurately can be implemented. As the heading is computed using the velocity components in the North and East directions, it is not accurate in cases

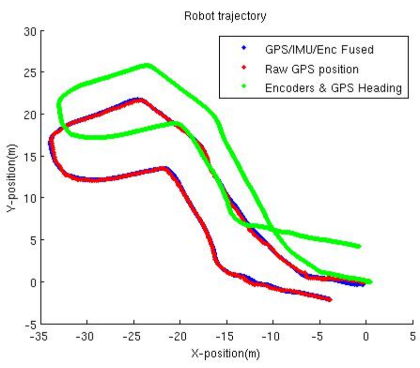

Fig. 4. Robot trajectories

where the robot just spins in place or moves very slowly. Also, a joint estimation of the robot's attitude and position can be done, similar to the method described in [4]. Instead of a UKF, a particle filter can be implemented. The particle filter would be more effective in resolving orientations compared to the existing UKF as it samples a much bigger space of orientations. We are currently working on this particle filter approach with process and measurement updates using the Unscented Transform (due to non-linear process updates).

IV. Operating Framework/State Machine

Any framework to assign states to a robot must reflect its goal and capabilities. Our main goal for the RASCAL rover is to explore the environment, detect and pick up objects of interest, doing so at the fastest possible time. There are a number of obstacles, either in the form of a rocky terrain or a moon crater and so on. Also, there is a possibility of human control. To combine all these and get the system working efficiently, we need a strong framework and established states that the robot can be in. We implement this in the form of states in a Finite State Machine. There are quite a few advantages to this approach. At any point of time, the robot's state is known and can be easily traced. The transitions between the states are also well defined and there can be no bad jumps. This is also very useful in our setting as it allows for quick and easy control.

A. *States*

Our implementation of the state machine has four primary states, two of which are simple wait states where the robot waits for a specific command. The remaining states are the key components of our system. The states, their actions and transitions are summarised below.

- Initial state
  - Robot starts off at this state.
  - Waits until the localization module computes a good pose estimate
  - Has a single transition over to the ”Wait state”
- Wait state
  - Robot waits till it gets a ”GOAL”

    - Goal can be specified from external computer or computed through a color classifier
    - User specified goal has highest priority so as to enable the user to control robot's actions
    - Robot transitions to "Scan state" once goal is reached
- Scan state
    - To reach the goal, robot needs a map and a path through the environment
    - First, robot uses a LIDAR mounted on a tilt platform to create a local map
    - Robot then plans on this map to generate a planned path
    - It then transitions over to the Trajectory Follower
- Follow state
    - Robot takes the planned path and tries to follow it
    - We plan to use a Trajectory Rollout type controller
    - Has not been implemented yet

The above mentioned states are the most important states. In addition to this, there are a few other simple states which command the robot to perform a specific operation such as going forward, turning, backing up and so on. Also, there are a number of state transitions that have not been mentioned above. A few are:

- The robot transitions to the Scan state as soon as it has received a User defined goal point irrespective of its current state(Except if it is in the Initial state)
- Once the robot has reached a goal point, it transitions to the Wait state
- In case of a timeout in the Scan or Follow states, the robot transitions to the Wait state
- In the Follow state, if the robot travels a distance greater than 2 meters, it transitions to the Scan state. This is to make sure that the robot has sufficient information from the map about possible obstacles on the ground.

Two main components of the Scan state deserve special mention, namely the Costmap generation and the Planner.

1) Costmap generation: The RASCAL robot is supposed to operate in an unknown environment and as such, does not know its layout. Even though there is a human in the loop who can help the robot avoid obstacles, it is best to create and store a map of the robot's environment. To serve this purpose, we have designed a simple method to create a costmap. The rover is equipped with a LIDAR on a tilting platform. This was chosen rather than a vertical LIDAR as most of the obstacles that we may encounter may be rocks rather than walls. Once a goal point is specified, the robot spins until it faces the direction of the goal. It then scans the environment by tilting the laser. Once these points are obtained, they are rotated and translated to the world frame using the robot's current pose estimate and the angle of the tilting platform. The 3D points are then projected onto the XY plane (Z is up). Points which belong to ground are isolated and their

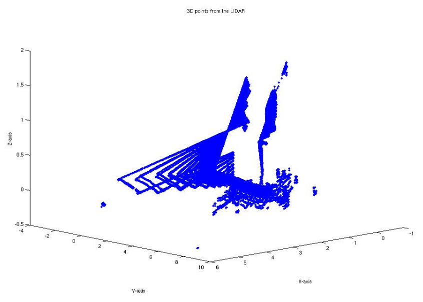

Fig. 5. 3D points from LIDAR on tilt platform

costs reduced while obstacles have a higher cost. Unexplored regions have a medium cost. The costmap so obtained is used by the planner to generate a path to the goal.

2) Motion Planner: Now the robot needs to plan a safe and traversable path to the goal from its current position. The obstacles are at first inflated by the robot's footprint and planning is done on this new costmap. An efficient implementation of the AStar search algorithm is used for this purpose. Euclidean distance is used as the heuristic. The implementation was done in C++ using a priority queue to efficiently sort the states. The planner is capable of operating close to real time even on maps of size 1000x1000.

B. Results and Discussion

The State Machine conceptualized above was developed and extensively tested indoors. Each of the individual components were tested and their performance was evaluated. Figures 5,6 and 7 show the 3D lidar points, the costmap and the costmap with inflated obstacles respectively for a particular test case. Note the scale on the X and Z axes in figure 5. Points on the costmap which have a low cost are shown in blue. The planned path is shown in red in figures 6 and 7. The test was conducted with the robot in a hallway with a door to its side. Goal positions were specified by an external user. From the costmap, it is clear that the robot is able to create an accurate enough map of the environment for path planning. All these were computed in real time on the robot with the various transitions listed as above. Due to the absence of a proper trajectory follower, no tests were conducted where the robot tried to reach the goal point. Most of the other states and their transitions were tested and found to be proper. Figures 8 and 9 show similar results.

C. Improvements

The performance of the State Machine must be tested outdoors with all the states assigned. A trajectory follower type controller can be implemented. The costmap generated is good only for local navigation. To overcome this, the rover can stop once every few meters and "refresh" its map of the environment. This would make the system more robust and would also prevent the robot from bumping into obstacles. Online obstacle detection may also be performed by fixing

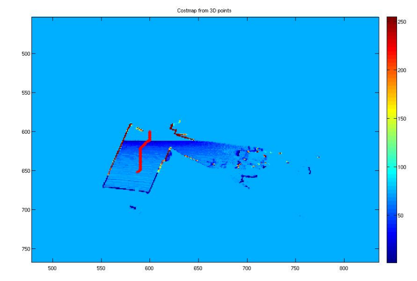

Fig. 6. Costmap from LIDAR, planned path in red

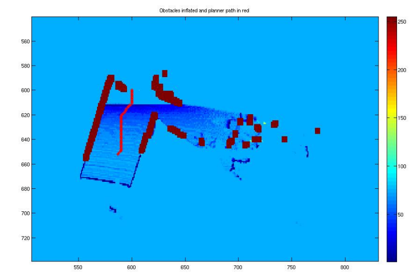

Fig. 7. Costmap with inflated obstacles, planned path in red

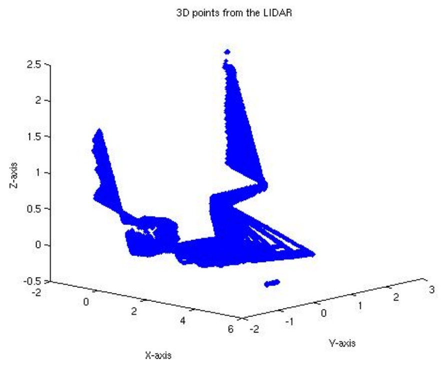

Fig. 8. 3D points from LIDAR on tilt platform

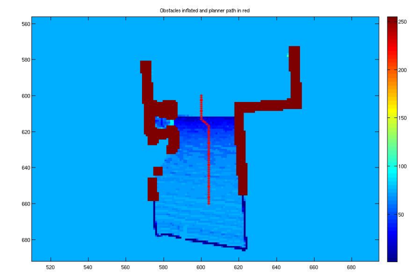

Fig. 9. Costmap with inflated obstacles, planned path in red

the LIDAR at a particular angle. Also, pose information may be integrated with the map to create a global map of the environment. Finally, the 3D LIDAR points can also be used to detect the slope of the ground and as such may serve as a ”ramp” detector.

V. Conclusion

In this work, we presented a simple structure for outdoor localization and navigation in unknown environments. An Unscented Kalman Filter based approach was used to track the robot's position over time. A state machine was implemented to limit the states of the robot and allow for easy transitions from one action to another. The subsystems were tested and their performances were evaluated. The components were found to be accurate and efficient at trying to solve the problem at hand. A few additions to the existing systems are also being developed to complete the system and further boost performance.

References

[1] F. Ababsa, *Advanced 3D Localization by Fusing Measurements from GPS, Inertial and Vision Sensors*, Autonomous Robots Vol.27, No.1, 2009.

[2] E. Kraft, *A Quaternion based Unscented Kalman Filter for Orientation Tracking*, ICIF Vol.1, 2003.

[3] S. J. Julier and J. K. Uhlmann, *A new extension of the Kalman Filter to Non-linear systems*, Int. Symp. Aerospace/Defense Sensing, Simul. and Controls, 1997.

[4] P. Vernaza and D. D. Lee, *Rao-Blackwellized Particle Filtering for 6-DOF Estimation of Attitude and Position via GPS and Inertial Sensors*, ICRA, 2006.

[5] G. Welch and G. Bishop, *An Introduction to the Kalman Filter*, Technical Report TR 95-041, University of North Carolina, Department of Computer Science, 1995